Micriμm

Empowering Embedded Systems

μC/OS-II

μC/Probe

and the

NXP LPC3250/LPC3180 Processors

(Using the phyCORE®-ARM9/LPC3250/LPC3180 Rapid Development Kit)

Application Note

AN-1250

www.Micrium.com

About Micriμm

Micriμm provides high-quality embedded software components in the industry by way of engineer-friendly source code, unsurpassed documentation, and customer support. The company's world-renowned real-time operating system, the Micriμm **μC/OS-II**, features the highest-quality source code available for today's embedded market. Micriμm delivers to the embedded marketplace a full portfolio of embedded software components that complement **μC/OS-II**. A TCP/IP stack, USB stack, CAN stack, File System (FS), Graphical User Interface (GUI), as well as many other high quality embedded components. Micriμm's products consistently shorten time-to-market throughout all product development cycles. For additional information on Micriμm, please visit www.micrium.com.

About μC/OS-II

Thank you for your interest in **μC/OS-II**. **μC/OS-II** is a preemptive, real-time, multitasking kernel. **μC/OS-II** has been ported to over 45 different CPU architectures and now, has been ported to the Phytec phyCORE®-ARM9/LPC3250 and phyCORE®-ARM9/LPC3180 Rapid Development Kit (RDK) evaluation boards which include the ARM-based NXP LPC3250 and LPC3180 respectively processors.

μC/OS-II is small yet provides all the services you would expect from an RTOS: task management, time and timer management, semaphore and mutex, message mailboxes and queues, event flags a much more.

You will find that **μC/OS-II** delivers on all your expectations and you will be pleased by its ease of use.

Licensing

μC/OS-II is provided in source form for **FREE** evaluation, for educational use or for peaceful research. If you plan on using **μC/OS-II** in a commercial product you need to contact Micriμm to properly license its use in your product. We provide ALL the source code with this application note for your convenience and to help you experience **μC/OS-II**. The fact that the source is provided **DOES NOT** mean that you can use it without paying a licensing fee. Please help us continue to provide the Embedded community with the finest software available. Your honesty is greatly appreciated.

About μC/Probe Demo Version

μC/Probe is a Windows application that allows a user to display and change the value (at run-time) of virtually any variable or memory location on a connected embedded target. The user simply populates **μC/Probe**'s graphical environment with gauges, tables, graphs, and other components, and associates each of these with a variable or memory location. Once the application is loaded onto the target, the user can begin **μC/Probe**'s data collection, which will update the screen with variable values fetched from the target.

μC/Probe retrieves the values of global variables from a connected embedded target and displays the values in an engineer-friendly format. The supported data-types are: booleans, integers, floats and ASCII strings.

μC/Probe can have any number of 'data screens' where these variables are displayed. This allows to logically grouping different 'views' into a product.

This **μC/Probe** demo version can only retrieve information from RS-232C or J-LINK interfaces and is limited up to 15 symbols.

The demo version of **μC/Probe** is available on the Micriμm website:

http://www.micrium.com/products/probe/probe.html

About μC/Probe Full Version

The full version of **μC/Probe** allows you to use a TCP/IP is a Windows application that allows a user to display and change the value (at run-time) of virtually any variable or memory location on a connected embedded target. The user simply populates **μC/Probe**'s graphical environment with gauges, tables, graphs, and other components, and associates each of these with a variable or memory location. Once the application is loaded onto the target, the user can begin **μC/Probe**'s data collection, which will update the screen with variable values fetched from the target.

Manual Versions

If you find any errors in this document, please inform us and we will make the appropriate corrections for future releases.

Version	Date	By	Description
V.1.00	2009/03/31	FT	Updated for Phytec LPC3250 rev 1305.3 board
V.1.00	2008/08/20	FT	Initial version.

Software Versions

This document may or may not have been downloaded as part of an executable file, *Micrium-NXP-uCOS-II–LPC3250-Phytec.exe*, containing the code and projects described here. If so, then the versions of the Micriμm software modules in the table below would be included. In either case, the software port described in this document uses the module versions in the table below

Module	Version	Comment
μC/OS-II	V2.88	
μC/OS-Probe	V2.30	

Document Conventions

Numbers and Number Bases

- Hexadecimal numbers are preceded by the "0x" prefix and displayed in a monospaced font. Example: `0xFF886633`.
- Binary numbers are followed by the suffix "b"; for longer numbers, groups of four digits are separated with a space. These are also displayed in a monospaced font. Example: `0101 1010 0011 1100b`.
- Other numbers in the document are decimal. These are displayed in the proportional font prevailing where the number is used.

Typographical Conventions

- Hexadecimal and binary numbers are displayed in a monospaced font.
- Code excerpts, variable names, and function names are displayed in a monospaced font. Functions names are always followed by empty parentheses (e.g., `OS_Start()`). Array names are always followed by empty square brackets (e.g., `BSP_Vector_Array[]`).
- File and directory names are always displayed in an italicized serif font. Example: */Micrium/Sofware/uCOS-II/Source/*.
- A bold style may be layered on any of the preceding conventions—or in ordinary text—to more strongly emphasize a particular detail.
- Any other text is displayed in a sans-serif font.

Table of Contents

1. Installing the Micirum Software 7

2. Using the Micirum IAR Example Project 8
2.01.01 IAR **μC/OS-II** Kernel Awareness. 8
2.01.02 IAR Project Configurations 9
2.02 Running the Example Applications 10

3. Directories and Files 13

4. Application Code 17

4.01 *app.c* 17
4.02 *os_cfg.h* 19

5. Board Support Package (BSP) 21
5.01 BSP, *bsp_xxx.c* and *bsp_xxx.h* files 21
5.02 Board Support Package Configuration 23

6. μC/Probe 24

Licensing 27

References 27

Contacts 27

1. Installing the Micirum Software

The source code for **μC/OS-II** is provided in source form along with IAR EW project files that allows you to run **μC/OS-II** on the Phytec LPC3250 and LPC3180 development boards. To install the software, simply run the self-extracting executable. ***Micrium-NXP-uCOS-II LPC3xxx-Phytec.exe.***

You will be prompted to accept the simple terms of the licensing agreement. If you answer 'Yes', the software will be installed on your PC under the ***\Micrium*** directory from the root as shown in Figure 1-1

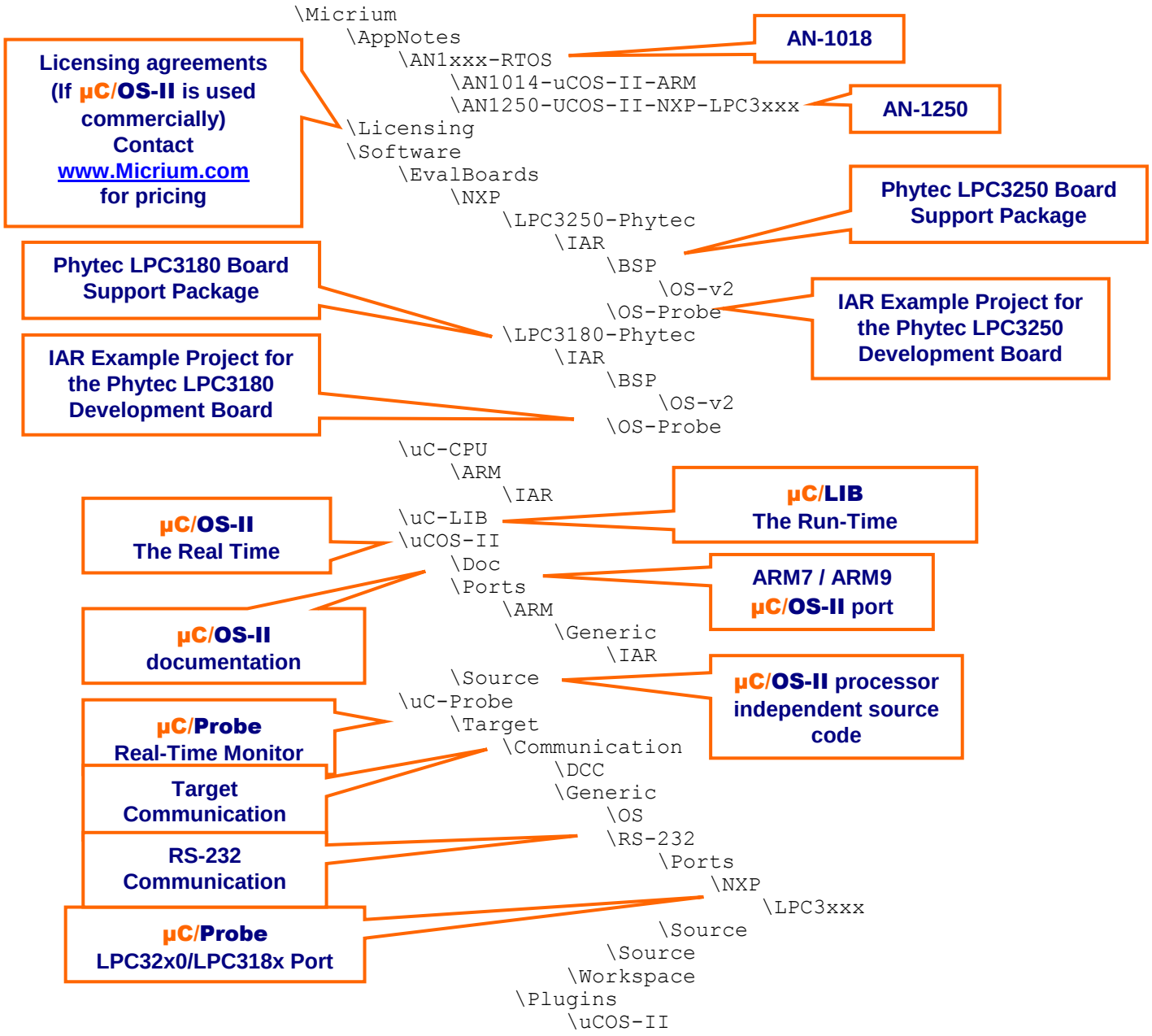

Figure 1-1. Directory Structure

2. Using the Micirum IAR Example Project

To view the example project, start an instance of IAR Embedded Workbench, and open:

- For the Phytec LPC3250 evaluation board , ***LPC3250-Phytec-OS-Probe.ewp***, located in ***/Micrium/Software/EvalBoards/NXP/LPC3250-Phytec/IAR/OS-Probe***

- For the Phytec LPC3180 evaluation board , ***LPC3180-Phytec-OS-Probe.ewp***, located in ***/Micrium/Software/EvalBoards/NXP/LPC3250-Phytec/IAR/OS-Probe***

To do this, use the *Add Existing Project...* menu command under the *Project* menu:

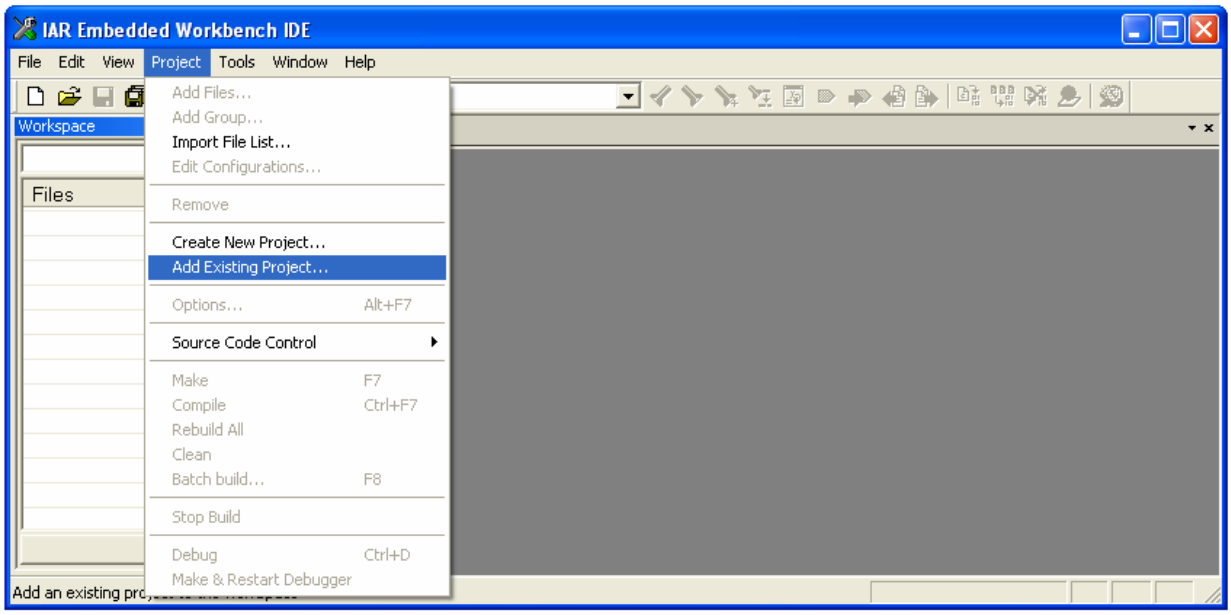

Figure 2-1. IAR EW. Opening an existing project

IAR EWARM Versions

Be certain to open the proper project for your version of EWARM. The Phytec LPC3250/PC3280 examples projects were created using **EWARM ver. 5.3**

2.01.01 IAR μC/OS-II Kernel Awareness.

The **μC/OS-II** Kernel Awareness plug-in will allow you to examine information about system objects while using the C-Spy debugger. To gain access to this feature, enable the plug-in by right-clicking on the project name in the work space browser and choosing *Options…* Then, select the "Debugger" entry in the list box, and the "Plug-in" tab pane. Find the **μC/OS-II** entry in the list and, finally, select the check box beside the entry.

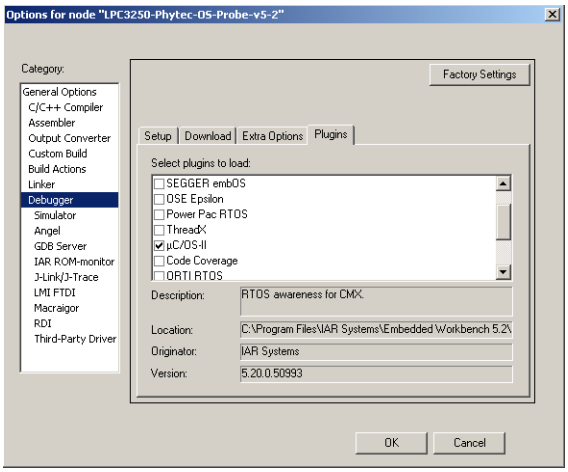

Figure 2-2. Enabling the μC/OS-II Kernel Awareness Plug-In

2.01.02 IAR Project Configurations

The IAR project configurations allow you to compile, link and load the software in different ways to the target. The following configurations are available in the IAR projects as shown in figure 2-3.

- IRAM: This project option is configured to load the code into the processor's internal RAM (IRAM).
- XRAM: This project option is configured to load the code into the board's external SDRAM (XRAM).
- IRAM boot from NAND: This option is configured to flash the code into the board's external NAND memory and load it to the IRAM using the example bootloader provided by IAR.
- XRAM boot from NAND: This option is configured to flash the code into the board's external NAND memory and load it to the XRAM using the example bootloader provided by IAR.

Note (1): In order to used the **"IRAM boot from NAND"** configuration in the Phytec LPC3250 project, you must copy the file ***FlashPhytecLPC3250_NAND_IRAMload.xml*** from the example project folder

/Micrium/Software/EvalBoards/NXP/LPC3250-Phytec/IAR/OS-Probe

to the IAR EW installation folder

\Program Files\IAR Systems\Embedded Workbench 5.4/arm/config/flashloader/NXP

Note (2): In order to used the **"IRAM boot from NAND"** configuration in the Phytec LPC3180 project, you must copy the file ***FlashPhytecLPC3180_NAND_IRAMload.xml*** from the example project folder

/Micrium/Software/EvalBoards/NXP/LPC3180-Phytec/IAR/OS-Probe

to the IAR EW installation folder

\Program Files\IAR Systems\Embedded Workbench 5.4/arm/config/flashloader/NXP

Phytec LPC3250 Only.

Note (3): Be certain to select the proper configuration for you revision of the board.

IRAM boot from NAND (32M)
XRAM boot from NAND (32 M)
Only can be used with revision 1304.0 of the Module board.

IRAM boot from NAND (64 M)
XRAM boot from NAND (64 M)
Only can be used with revision 1304.1 of the Module board.

Phytec LPC31250 Module Board ver. 1304.0

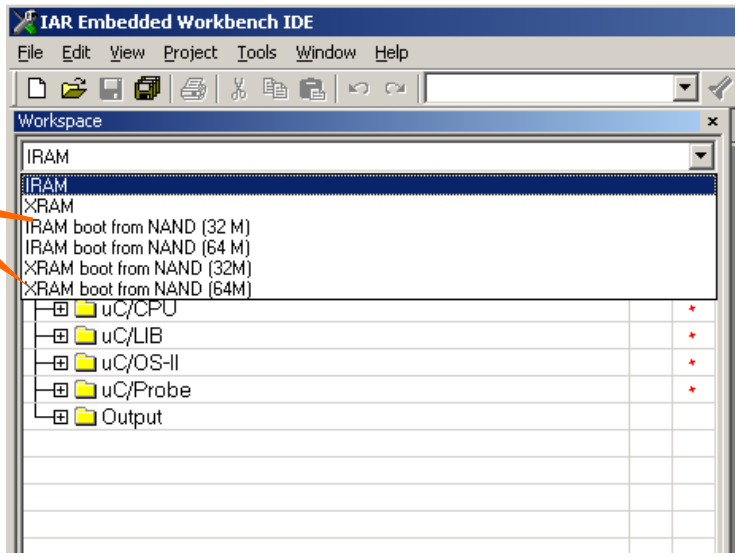

Figure 2-3. IAR project configurations

2.02 Running the Example Applications

The example project includes a basic demonstration of **μC/OS-II** and **μC/Probe.** The evaluation board components are labeled in the figures 2-3 and 2-4:

The example application is the same for the both Phytec boards.

To communicate with the board through RS-232, connect a serial cable between the evaluation board serial port "BTM=UART5" and your PC and open a HyperTerminal or any Terminal program window. Configure the RS-232 interface with the following settings:

- **Bits per Second**: 115200
- **Data bits**: 8
- **Parity:** None
- **Stops bits:** 1
- **Flow Control:** None

Once the program is loaded onto the target, the LEDs will start blinking.

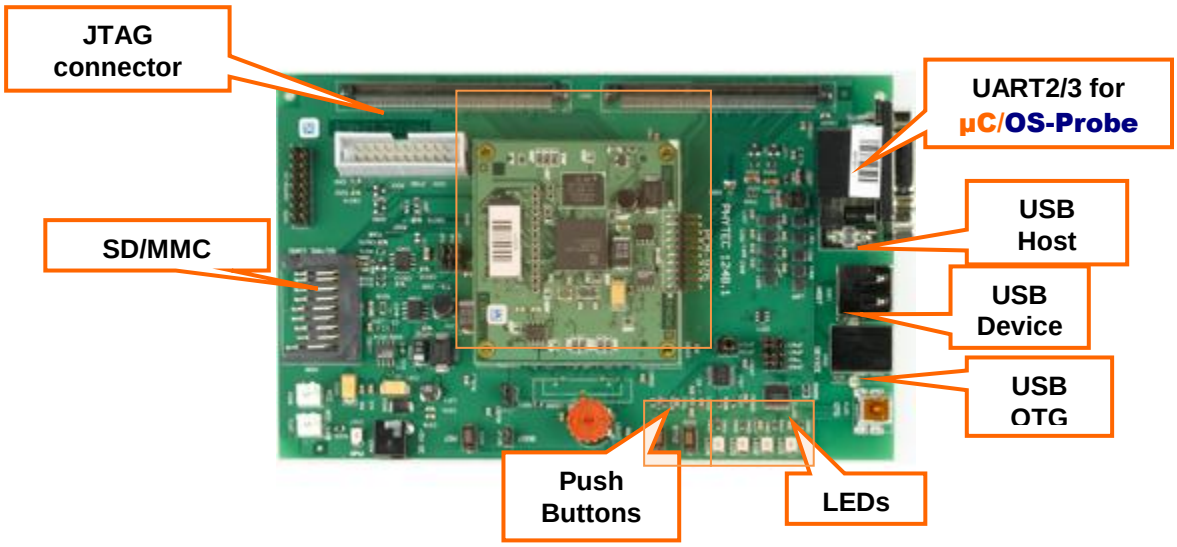

Figure 2-3. Phycore-LPC3180 Rapid Development Kit (RDK)

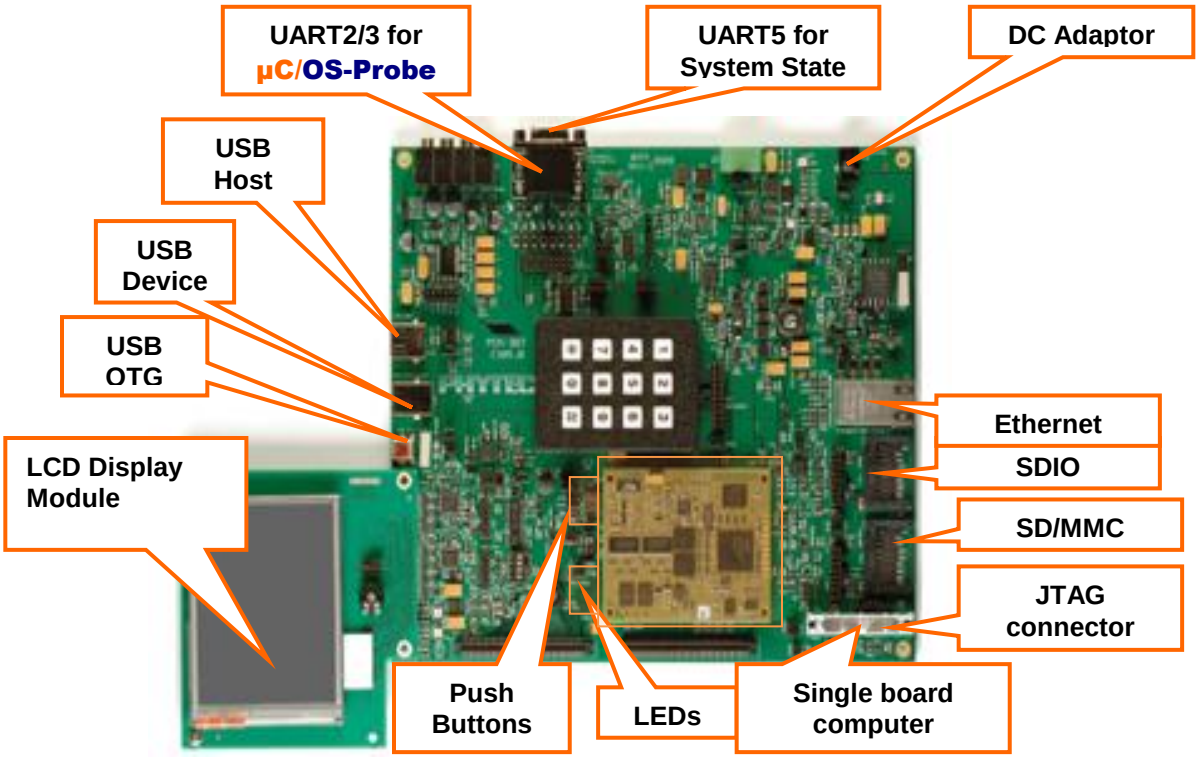

Figure 2-4. Phycore-LPC3250 Rapid Development Kit (RDK)

The lower RS-232 port labeled as “BTM=UART5” (at 115200 baud) is used to output text.

The upper RS-232 port labeled as “TOP=UART 3/2” is used for **μC/Probe** (at 115200 baud), which allows you to view (in real-time) the value of static / global variables in the target system.

The text shown in Figure 2-5 will be displayed in the HyperTerminal Window.

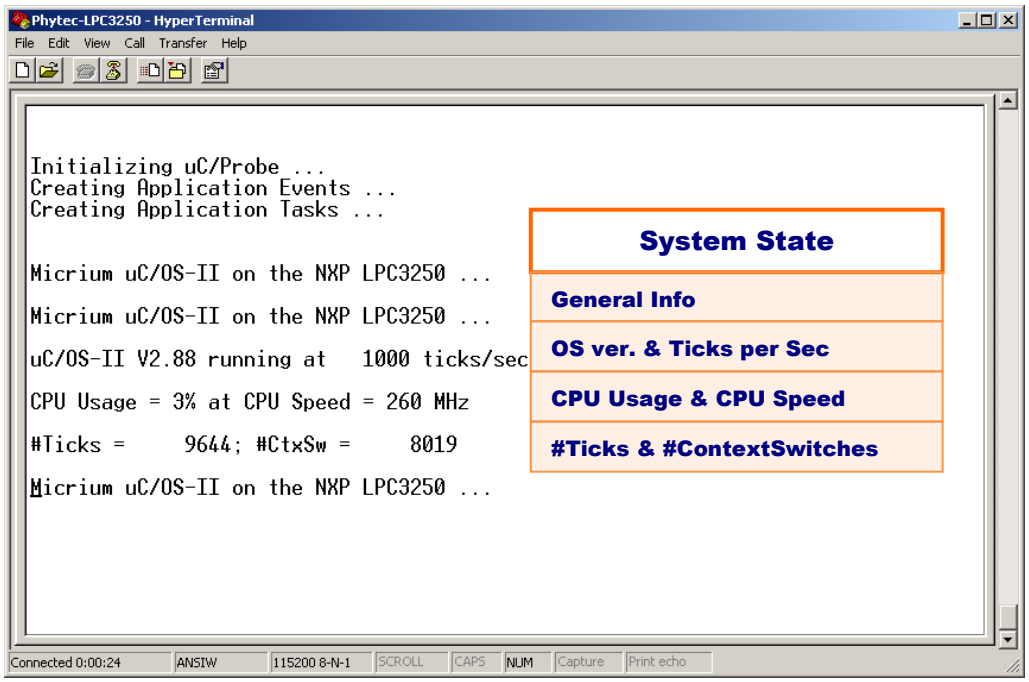

Figure 2-5. Application output (LPC3250)

The push buttons can be used to change the output to a new item.

If you use the “IRAM/XRAM boot from NAND configuration” you might have to stop the debugger first and power cycle the board.

3. Directories and Files

Application Notes

\Micrium\AppNotes\AN1xxx-RTOS\AN1014-uCOS-II-ARM

This directory contains ***AN-1014.pdf***, the application note describing the ARM port for **μC/OS-II**, and ***AN-1014-PPT.pdf***, a supplement to *AN-1014.pdf*.

\Micrium\AppNotes\AN1xxx-RTOS\AN1250--uCOS-II-NXP-LPC3xxx

This directory contains this application note, ***AN-1250.pdf***.

Licensing Information

\Micrium\Licensing

Licensing agreements are located in this directory. Any source code accompanying this appnote is provided for evaluation purposes only. If you choose to use **μC/OS-II** in a commercial product, you must contact Micriμm regarding the necessary licensing.

μC/OS-II Files

\Micrium\Software\uCOS-II\Doc

This directory contains documentation for **μC/OS-II**.

\Micrium\Software\uCOS-II\Ports\ARM\Generic\IAR

This directory contains the standard processor-specific files for the generic **μC/OS-II** ARM port assuming the IAR toolchain and the ARM/Keil "RealView Microprocessor Development Kit" toolchain. These files could easily be modified to work with other toolchains (i.e., compiler/assembler/linker/locator/debugger); however, the modified files should be placed into a different directory. The following files are in this directory:

- *os_cpu.h*
- *os_cpu_a.asm*
- *os_cpu_c.c*
- *os_dcc.c*
- *os_dbg.c*

With this port, **μC/OS-II** can be used in either ARM or Thumb mode. Thumb mode, which drastically reduces the size of the code, was used in this example, but compiler settings may be switched (as discussed in Section 2.30) to generate ARM-mode code without needing to change either the port or the application code. The ARM/Thumb port is described in application note *AN-1014* which is available from the Micrium web site.

\Micrium\Software\uCOS-II\Source

This directory contains the processor-independent source code for **μC/OS-II**.

μC/Probe Files

\Micrium\Software\uC-Probe\Communication\Generic

This directory contains the **μC/Probe** generic communication module, the target-side code responsible for responding to requests from the **μC/Probe** Windows application (including requests over RS-232).

\Micrium\Software\uC-Probe\Communication\Generic\Source

This directory contains ***probe_com.c*** and ***probe_com.h***, the source code for the generic communication module.

\Micrium\Software\uC-Probe\Communication\Generic\OS\uCOS-II

This directory contains ***probe_com_os.c***, which is the **μC/OS-II** port for the **μC/Probe** generic communication module.

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232

This directory contains the RS-232 specific code for **μC/Probe** generic communication module, the target-side code responsible for responding to requests from the **μC/Probe** Windows application over RS-232

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232\Source

This directory contains ***probe_rs232.c*** and ***probe_rs232.h***, the source code for the generic communication module RS-232 code.

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232\Ports\NXP\LPC3xxx

This directory contains ***probe_rs232c.c*** and ***probe_rs232c.h***, the NXP LPC3xxx port for the RS-232 communications.

\Micrium\Software\uC-Probe\Communication\Generic\Source\RS-232\OS\uCOS-II

This directory contains ***probe_rs232_os.c***, which is the **μC/OS-II** port for the **μC/Probe** RS-232 communication module.

μC/CPU Files

\Micrium\Software\uC-CPU

This directory contains ***cpu_def.h***, which declares `#define` constants for CPU alignment, endianness, and other generic CPU properties.

\Micrium\Software\uC-CPU\ARM\IAR

This directory contains ***cpu.h*** and ***cpu_a.s***. *cpu.h* defines the Micriμm portable data types for 8-, 16-, and 32-bit signed and unsigned integers (such as `CPU_INT16U`, a 16-bit unsigned integer). These allow code to be independent of processor and compiler word size definitions. *cpu_a.s* contains generic assembly code for ARM7 and ARM9 processors which is used to enable and disable interrupts within the operating system. This code is called from C with `OS_ENTER_CRITICAL()` and `OS_EXIT_CRITICAL()`.

μC/LIB Files

\Micrium\Software\uC-LIB

This directory contains ***lib_def.h***, which provides `#defines` for useful constants (like `DEF_TRUE` and `DEF_DISABLED`) and macros.

\Micrium\Software\uC-LIB\Doc

This directory contains the documentation for **μC/LIB**.

Application Code

\Micrium\Software\EvalBoards\NXP\LPC3180-Phytec\IAR\OS-Probe
\Micrium\Software\EvalBoards\NXP\LPC3250-Phytec\IAR\OS-Probe

This directory contains the source code the example application:

- ***app.c*** contains the test code for the example application including calls to the functions that start multitasking within **μC/OS-II**, register tasks with the kernel, and update the user interface (the LEDs and the push buttons).
- ***app_cfg.h*** is a configuration file specifying stack sizes and priorities for all user tasks and `#define`s for important global application constants.
- ***app_probe.c/h*** contain code to initialize **μC/Probe**,
- ***app_hooks.c/h*** contain code for the **μC/OS-II** application hooks.
- ***includes.h*** is the master include file used by the application.
- ***os_cfg.h*** is the **μC/OS-II** configuration file.
- ***probe_com_cfg.h*** is the **μC/Probe** configuration file.
- ***LPC3250-Phytec-OS-Probe.**** are the IAR Embedded Workbench project files for the Phytec LPC3250 Board.
- ***LPC3180-Phytec-OS-Probe.**** are the IAR Embedded Workbench project files for the Phytec LPC3180 board.

\Micrium\Software\EvalBoards\NXP\LPC3180-Phytec\IAR\BSP
\Micrium\Software\EvalBoards\NXP\LPC3250-Phytec\IAR\BSP

This directory contains the Board Support Package for the PHYCore-LPC3250 RDK:

- ***bsp.c /bsp.h*** contain generic BSP functions which initialize critical processor functions (e.g., the PLL) and provide support for peripherals such as the push button and LEDs.
-
- ***bsp_a.s*** contains the low level functions (written in assembly language) for the PHYCore-LPC3250 RDK
- ***bsp_int.c/bsp_int.h*** contains functions related to the interrupt controller
- ***bsp_lcd.c/bsp_lcd.h*** contains functions related to the LCD controller (LPC3250 only).
- ***bsp_pwr_ctrl.c/bsp_pwr_ctrl.h*** contains functions related to Power & clock controller .
- ***bsp_touch_scr.c/bsp_touch_scr.h*** contains functions related to the touch screen controller LPC3250 (only)

- ***bsp_ser.c/bsp_ser.h*** contains functions related to UARTs.

- ***cstartup.s*** performs critical processor initialization (such as the initialization of task stacks), readying the platform to enter `main()`.

- ***LPC3250_xRAM.icf/LPC3180_xRAM.icf*** is the IAR linker files which contain information about the placement of data and code segments in the processor's memory map, where 'x' can be 'I' for internal RAM (IRAM) or 'X' for external RAM (XRAM).

- ***LPC3250_xRAM.mac/LPC3180_xRAM.mac*** contains instructions that are executed prior to loading code onto the processor, where 'x' can be 'I' for internal RAM (IRAM) or 'X' for external RAM (XRAM)

Micrium\Software\EvalBoards\NXP\LPC3180-Phytec\IAR\BSP\OS-v2
Micrium\Software\EvalBoards\NXP\LPC3250-Phytec\IAR\BSP\OS-v2

- ***bsp_os.c/h*** encapsulates some of the **μC/OS-II** kernel services. Implements the timer initialization needed for the ticker interrupt needed in **μC/OS-II** .

4. Application Code

The example application described in this appnote, *AN-1250*, is a simple demonstration of **μC/OS-II** and **μC/OS-Probe** for the NXP LPC3250 processor on the PHYCore-LPC3250 RDK and the NXP LPC3180 LPC3250 processor on the PHYCore-LPC3180 RDK.

4.01 *app.c*

Four functions of interest are located in *app.c*:

1. **`main()`** is the entry point for the application, as it is with most C programs. This function initializes the operating system, creates the primary application task, `App_TaskStart()`, begins multitasking, and exits.

2. **`App_TaskStart()`**, after creating the application events and tasks, enters an infinite loop in which it blinks the LEDs.

3. **`App_TaskKbd()`** polls the user inputs—the push buttons—and, if new input is detected, places a message in a queue for `App_TaskSer()`.

4. **`App_TaskUserIF()`**,Outputs the state of the system based on the display state passed to it by `App_TaskKbd()`.

```
int  main (void)                                                    /* Note 1 */
{
#if (OS_TASK_NAME_EN > 0)
    CPU_INT08U  err;
#endif


    BSP_IntDisAll();                                                /* Note 2 */

    OSInit();                                                       /* Note 3 */

    OSTaskCreateExt((void (*)(void *)) App_TaskStart,               /* Note 4 */
                    (void           *) 0,
                    (OS_STK         *)&AppTaskStartStk[APP_CFG_TASK_START_STK_SIZE - 1],
                    (INT8U           ) APP_CFG_TASK_START_PRIO,
                    (INT16U          ) APP_CFG_TASK_START_PRIO,
                    (OS_STK         *)&AppTaskStartStk[0],
                    (INT32U          ) APP_CFG_TASK_START_STK_SIZE,
                    (void           *) 0,
                    (INT8U           )(OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));

#if (OS_TASK_NAME_EN > 0)                                          /* Note 5 */
    OSTaskNameSet(APP_CFG_TASK_START_PRIO, (CPU_INT08U *)"Startup", &err);
#endif

    OSStart();                                                      /* Note 6 */
}
```

Listing 4-1, `main()`

Listing 4-1, Note 1: As with most C applications, the code starts in `main()`.

Listing 4-1, Note 2: All interrupts are disabled to make sure the application does not get interrupted until it is fully initialized.

Listing 4-1, Note 3: `OSInit()` must be called before creating a task or any other kernel object, as must be done with all **μC/OS-II** applications.

Listing 4-1, Note 4: At least one task must be created (in this case, using `OSTaskCreateExt()` to obtain additional information about the task). In addition, **μC/OS-II** creates either one or two internal tasks in `OSInit()`. **μC/OS-II** always creates an idle task, `OS_TaskIdle()`, and will create a statistic task, `OS_TaskStat()` if you set `OS_TASK_STAT_EN` to 1 in *os_cfg.h*.

Listing 4-1, Note 5: You can name **μC/OS-II** tasks (and other kernel objects) and display task names at run-time or with a debugger. In this case, the `App_TaskStart()` is given the name "Start Task". Because C-Spy can work with the Kernel Awareness Plug-In available from Micriμm, task names can be displayed during debugging.

Listing 4-1, Note 6: Finally multitasking under **μC/OS-II** is started by calling `OSSTart()`. **μC/OS-II** will then begin executing `App_TaskStart()` since that is the highest-priority task created (both `OS_TaskStat()` and `OS_TaskIdle()` having lower priorities).

```
static  void  App_TaskStart (void *p_arg)
{
    (void)p arg;

    BSP_Init();                                             /* Note 1 */

    OS_CPU_InitExceptVect();                                /* Note 2 */

    BSP_OS_TmrTickInit(OS_TICKS_PER_SEC);                   /* Note 3 */


#if OS TASK STAT EN > 0
    OSStatInit();                                           /* Note 4 */
#endif

    BSP_Ser_Init(115200);                                   /* Note 5 */
    APP TRACE INFO(("\n\n\r"));

#if (APP_CFG_PROBE_COM_EN       == DEF_ENABLED) || \
    (APP CFG PROBE OS PLUGIN EN == DEF ENABLED)
    App_ProbeInit();                                        /* Note 6 */
#endif

    BSP_LED_Off(0);

    APP TRACE INFO(("Creating Application Events ... \n\r"));
    App_EventCreate();                                      /* Note 7 */

    APP TRACE INFO(("Creating Application Tasks ... \n\r"));
    App_TaskCreate();

    while (DEF_TRUE) {                                      /* Note 8 */
        BSP_LED_Toggle(0);
        OSTimeDlyHMSM(0, 0, 0, 250);
        BSP LED Toggle(0);
        OSTimeDlyHMSM(0, 0, 0, 250);
    }
}
```

Listing 4-2, `App_TaskStart ()`

Listing 4-2, Note 1: `BSP_Init()` initializes the Board Support Package—the I/Os, LEDs, etc. See Section 5 for details.

Listing 4-2, Note 2: `OS_CPU_InitExceptVect()` Initializes the CPU interrupt vectors in RAM

Listing 4-2, Note 3: `BSP_OS_TmrTickInit()` Initializes the tick interrupt

Listing 4-2, Note 4: `OSStatInit()` initializes **μC/OS-II**'s statistic task. This only occurs if you enable the statistic task by setting `OS_TASK_STAT_EN` to 1 in *os_cfg.h*. The statistic task measures overall CPU usage (expressed as a percentage) and performs stack checking for all the tasks that have been created with `OSTaskCreateExt()` with the stack checking option set.

Listing 4-2, Note 5: **`BSP_Ser_Init()`** Initializes the RS-232 communication port at 115200.

Listing 4-2, Note 6: If **μC/OS-Probe** is enabled, then the module's initialization procedure **`App_ProbeInit()`** is called. **`App_ProbeInit()`** calls `OSProbe_Init()` which initializes the **μC/Probe** plug-in for **μC/OS-II**, which maintains CPU usage statistics for each task, `ProbeCom_Init()` that initializes the **μC/Probe** generic communication module and `ProbeRS232_Init()` that initializes the RS-232 communication module. After these have been initialized, the **μC/Probe** Windows program will be able to download data from the processor. For more information, see Section 6.

Listing 4-2, Note 7: **`App_EventCreate()`** Creates all the application uC/OS-II events and **`App_TaskCreate()`** creates all the application tasks.

Listing 4-2, Note 8: Any task managed by **μC/OS-II** must either enter an infinite loop 'waiting' for some event to occur or terminate itself. This task enters an infinite loop in which it toggles the LEDs .

4.02 *os_cfg.h*

The file ***os_cfg.h*** is used to configure **μC/OS-II** and defines the maximum number of tasks that your application can have, which services will be enabled (semaphores, mailboxes, queues, etc.), the size of the idle and statistic task and more. In all, there are about 60 or so `#define` that you can set in this file. Each entry is commented and additional information about the purpose of each `#define` can be found in Jean Labrosse's book, ***μC/OS-II, The Real-Time Kernel, 2nd Edition***. *os_cfg.h* assumes you have **μC/OS-II** V2.83 or higher but also works with previous versions of **μC/OS-II**.

- Task sizes for the Idle (**`OS_TASK_IDLE_STK_SIZE`**), statistics **`OS_TASK_STAT_STK_SIZE`**) and timer (**`OS_TASK_TMR_STK_SIZE`**) task are set to 128 `OS_STK` elements (each is 4 bytes) and thus each task stack is 512 bytes. If you add code to the examples make sure you account for additional stack usage.

- **`OS_DEBUG_EN`** is set to 1 to provide valuable information about **μC/OS-II** objects to IAR's C-Spy through the Kernel Awareness plug-in. Setting `OS_DEBUG_EN` to 0 should some code space (though it will not save much).

- **`OS_LOWEST_PRIO`** is set to 63, allowing up to 64 total tasks.

- **`OS_MAX_TASKS`** determines the number of "application" tasks and is currently set to 20 allowing 13 more tasks to be added to the example code.

- **OS_TICKS_PER_SEC** is set to 1000 Hz. This value can be changed as needed and the proper tick rate will be adjusted when the `BSP_OS_TmrTickInit()` is called. if you change this value. You would typically set the tick rate between 10 and 1000 Hz. The higher the tick rate, the more overhead **μC/OS-II** will impose on the application. However, you will have better tick granularity with a higher tick rate.

5. Board Support Package (BSP)

The Board Support Package (BSP) provides functions to encapsulate common I/O access functions and make porting your application code easier. Essentially, these files are the interface between the application and the PHYCore-LPC3250/LPC3180 RDK.

5.01 BSP, *bsp_xxx.c* and *bsp_xxx.h* files

Figure 5-1 and 5-2 shows the relationship between the BSP's files and the most important components on the development boards (not all functions are listed only the most relevants).

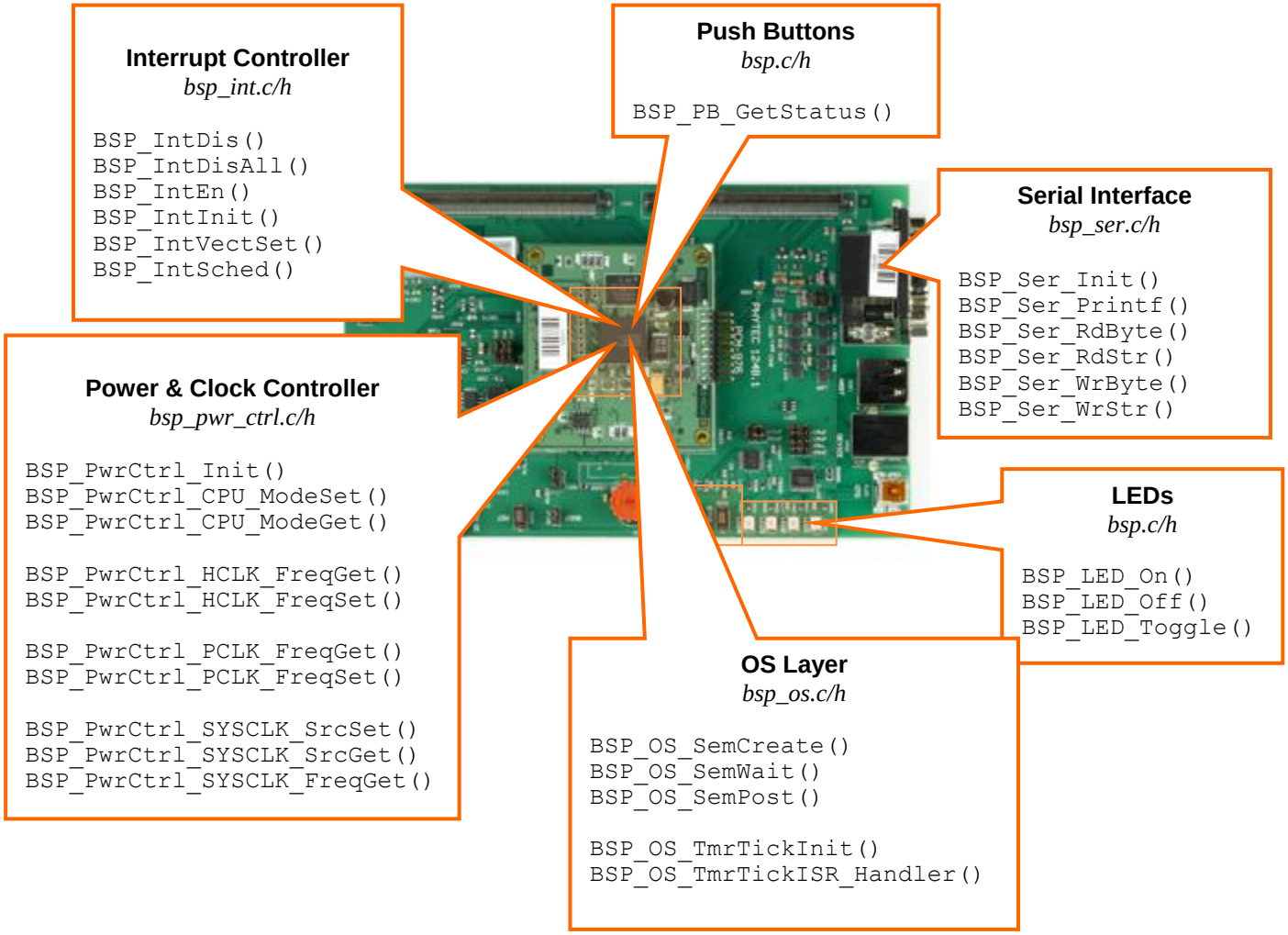

Figure 5-1. BSP functions for the Phycore-LPC3150 RDK

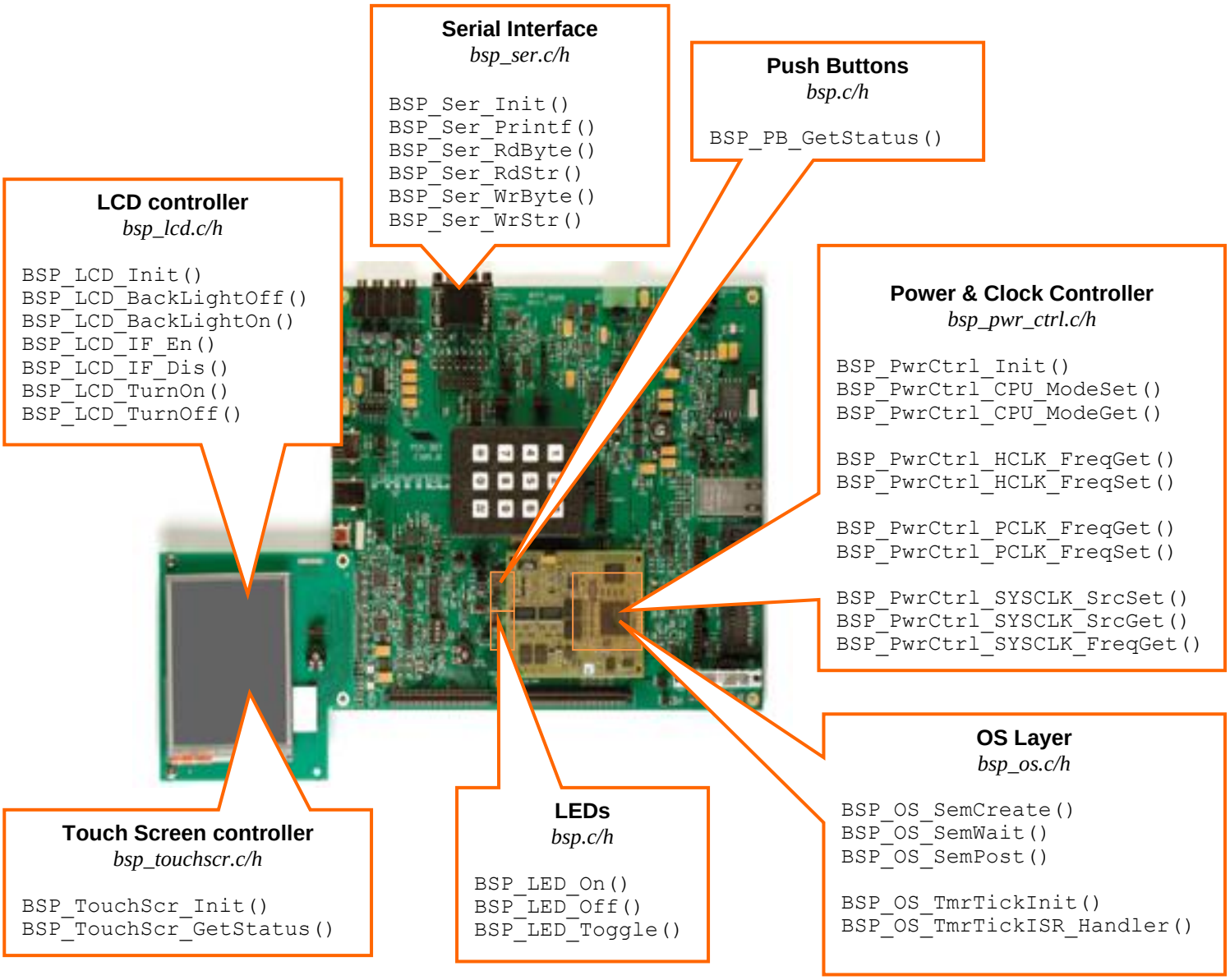

Figure 5-2. BSP functions for the Phycore-LPC3250 RDK

5.02 Board Support Package Configuration

Different aspects of the BSP can be configured at compile-time, allowing the user to select the version of the board, the serial port used to output the system state or the timer used for the tick interrupt:

`BSP_CFG_SER_COMM_SEL` Defines the serial port used to output the system state.

`BSP_SER_COMM_UART_05`	UART # 5
`BSP_SER_COMM_UART_03`	UART # 3 (Phytec LPC3250 only)
`BSP_SER_COMM_UART_02`	UART # 2 (Phytec LPC3180 Only)

`BSP_CFG_SER_TMR_SEL` Defines the Match register used in the High speed timer module (0, 1, or 2) for the tick Interrupt.

`BSP_CFG_SOM_VER` Defines the version of the module board (Phytec LPC3250 only)

`BSP_SOM_VER_0`	Phytec Module board ver. 1304.0
`BSP_SOM_VER_0`	Phytec Module board ver. 1304.1

`BSP_CFG_RDK_VER` Defines the version of the Rapid development kit (Phytec LPC3250 Only)

`BSP_RDK_VER_0`	Phytec Module board ver. 1305.0
`BSP_RDK_VER_1`	Phytec Module board ver. 1305.1
`BSP_RDK_VER_2`	Phytec Module board ver. 1305.2
`BSP_RDK_VER_3`	Phytec Module board ver. 1305.3

6. μC/Probe

μC/Probe is a Windows program which retrieves the values of global variables from a connected embedded target and displays the values in a engineer-friendly format. To accomplish this, an ELF file, created by the user's compiler and containing the names and addresses of all the global symbols on the target, is monitored by **μC/Probe**. The user places components (such as gauges, labels, and charts) into a Data Screen in a **μC/Probe** workspace and assigns each one of these a variable from the Symbol Browser, which lists all symbols from the ELF file. The symbols associated with components placed on an open Data Screen will be updated after the user presses the start button (assuming the user's PC is connected to the target).

A small section of code resident on the target receives commands from the Windows application and responds to those commands. The commands ask for a certain number of bytes located at a certain address, for example, "Send 16 bytes beginning at 0x0040102C". The Windows application, upon receiving the response, updates the appropriate component(s) on the screens with the new values.

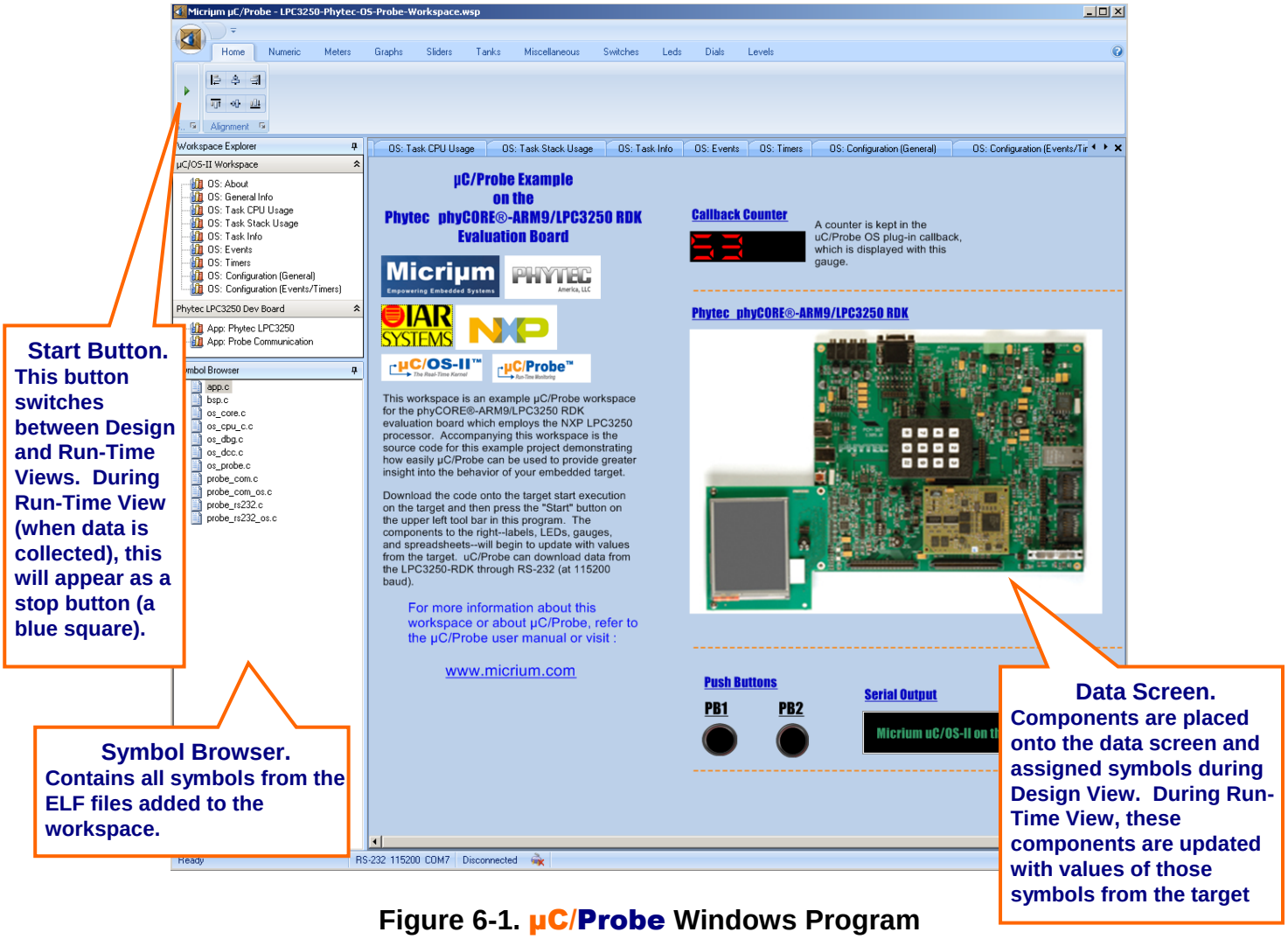
Start Button. This button switches between Design and Run-Time Views. During Run-Time View (when data is collected), this will appear as a stop button (a blue square).
Symbol Browser. Contains all symbols from the ELF files added to the workspace.
Data Screen. Components are placed onto the data screen and assigned symbols during Design View. During Run-Time View, these components are updated with values of those symbols from the target
μC/Probe Example on the Phytec phyCORE®-ARM9/LPC3250 RDK Evaluation Board
Callback Counter
A counter is kept in the uC/Probe OS plug-in callback, which is displayed with this gauge.
Phytec phyCORE®-ARM9/LPC3250 RDK
Push Buttons
PB1
PB2
Serial Output
For more information about this workspace or about μC/Probe, refer to the μC/Probe user manual or visit :
www.micrium.com

Figure 6-1. μC/Probe Windows Program

To use **μC/Probe** with the example project (or your application), do the following:

1. **Download and Install μC/Probe.** A trial version of **μC/Probe** can be downloaded from the Micriμm website at

   http://www.micrium.com/products/probe/probe.html

1. **Open μC/Probe**. After downloading and installing this program, open the example **μC/Probe** workspace for **μC/OS-II**, named ***OS-Probe-Workspace.wsp***, which should be located in your installation directory at

   /Program Files//Micrium/uC-Probe/Target/Plugins/uCOS-II/Workspace

2. **Connect Target to PC**. Currently, **μC/Probe** can use RS-232 to retrieve information from the target. You should connect a RS-232 cable between your target and computer.

3. **Load Your ELF File**. The example projects included with this application note are already configured to output an ELF file. (If you are using your own project, please refer to Appendix A of the **μC/Probe** user manual for directions for generating an ELF file with your compiler.) This file should be in

   /<Project Directory>/<Configuration Name>/exe/

   where <***Project Directory***> is the directory in which the IAR EWARM project is located (extension *.ewp) and <***Configuration Name***> is the name of the configuration in that project which was built to generate the ELF file and which will be loaded onto the target. The ELF file will be named

   <***Project Name***>***.elf***

   in EWARM v4.4x and

   <***Project Name***>***.out***

   in EWARM v5.xx unless you specify otherwise. To load this ELF file, right-click on the symbol browser and choose “Add Symbols”.

4. **Configure the RS-232 Options**. In **μC/Probe**, choose the “Options” menu item on the “Tools” menu. A dialog box as shown in Figure 6-2 (left) should appear. Choose the “RS-232” radio button. Next, select the “RS-232” item in the options tree, and choose the appropriate COM port and baud rate. The baud rate for the projects accompanying this appnote is 115200.
5. **Start Running**. You should now be ready to run **μC/Probe**. Just press the run button ( ) to see the variables in the open data screens update. Figure 6-3 displays the **μC/OS-II** workspace which displays detailed information about each task’s state.

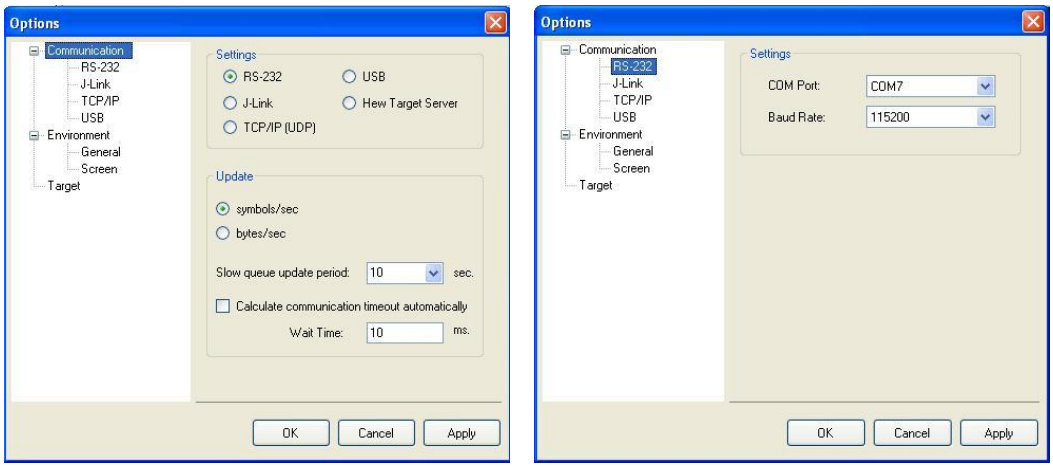

Figure 6.2. μC/Probe Options

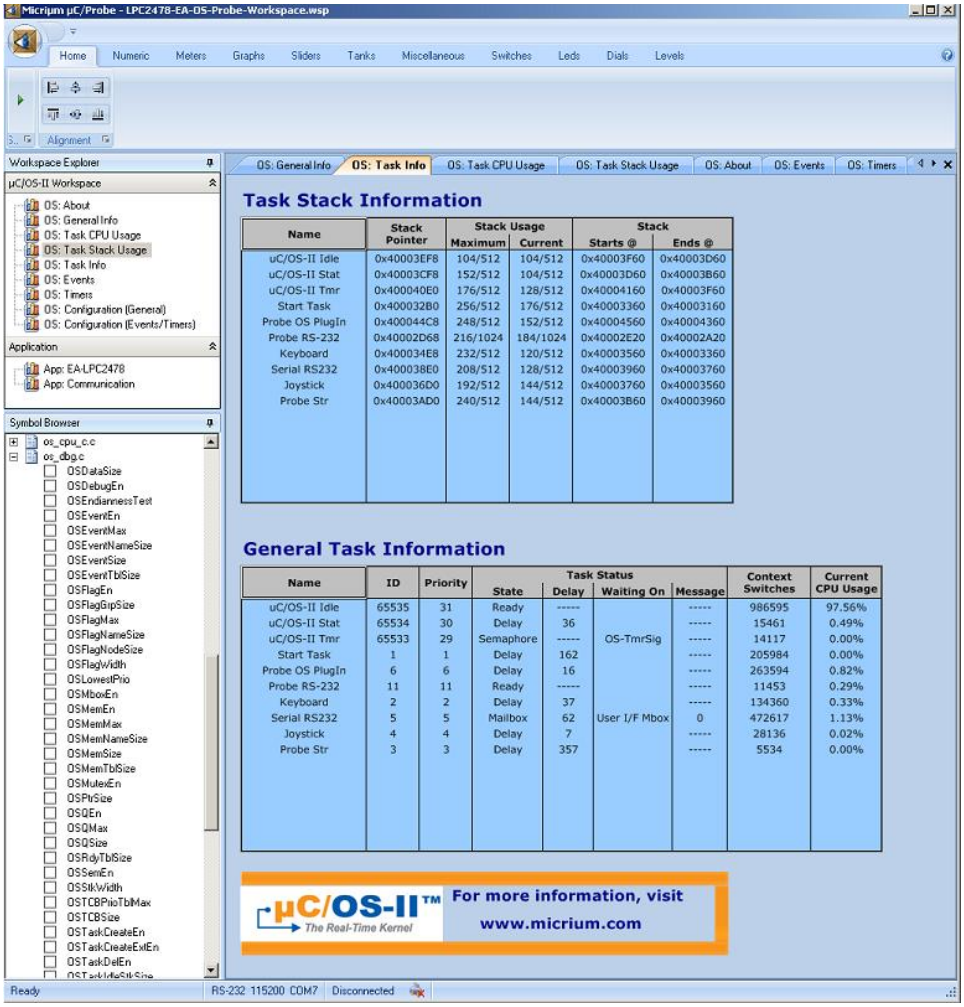

Figure 6-3. μC/Probe Run-Time: μC/OS-II Task Information

Licensing

µC/OS-II is provided in source form for **FREE** evaluation, for educational use or for peaceful research. If you plan on using **µC/OS-II** in a commercial product you need to contact Micriμm to properly license its use in your product. We provide **ALL** the source code with this application note for your convenience and to help you experience **µC/OS-II**. The fact that the source is provided does **NOT** mean that you can use it without paying a licensing fee. Please help us continue to provide the Embedded community with the finest software available. Your honesty is greatly appreciated.

References

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks
Jean J. Labrosse
R&D Technical Books, 2000
ISBN 0-87930-604-1

Contacts

IAR Systems
Century Plaza
1065 E. Hillsdale Blvd
Foster City, CA 94404
USA

+1 650 287 4250
+1 650 287 4253 (FAX)

e-mail: Info@IAR.com
WEB : www.IAR.com

CMP Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
USA

+1 785 841 1631
+1 785 841 2624 (FAX)

e-mail: rushorders@cmpbooks.com
WEB : http://www.cmpbooks.com

Micriμm
949 Crestview Circle
Weston, FL 33327
USA

+1 954 217 2036
+1 954 217 2037 (FAX)

e-mail: Jean.Labrosse@Micrium.com
WEB : www.Micrium.com

NXP
1110 Ringwood Court
San Jose, CA 95131
USA

+1 408 474 8142

WEB : www.nxp.com